

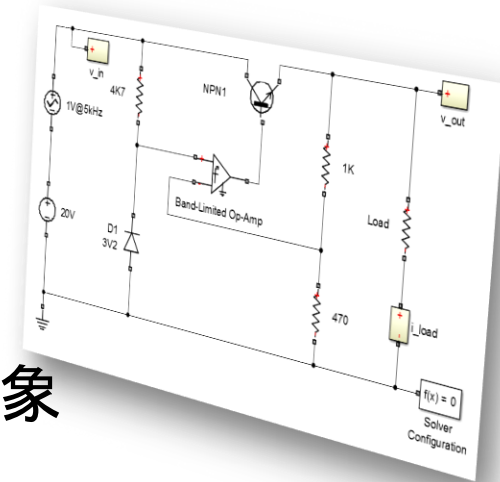
モデルを活用した パワーエレクトロニクス制御開発： 実装/試験編

MathWorks Japan
アプリケーションエンジニアリング部
アプリケーションエンジニア
新井 克明

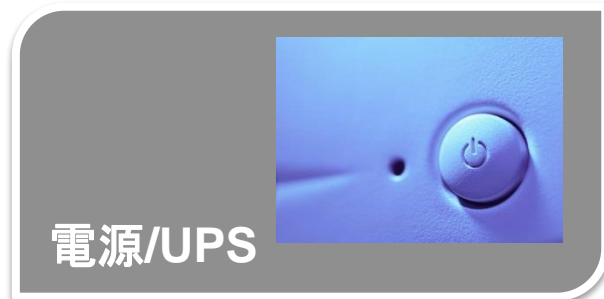
パワーエレクトロニクス制御と広がる活用分野

◆ パワーエレクトロニクス制御

- 電力の有効・安全利用のために
電流・電圧を所望の形に変換/制御
- 直流/交流/電圧/電流/周波数/位相が制御対象



◆ 活用分野



パワーエレクトロニクスに対するニーズ

◆ 限られた開発リソース/機能的制約下での付加価値の創出

■ 高効率化

・環境規制対策

(e.g. 省エネ法 / 国際エネルギースタープログラム)

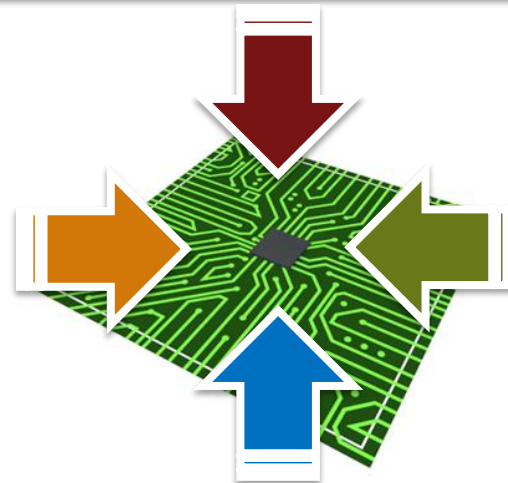
・動作時/待機時の省電力の両立

■ 高品質化

・低リップル

・デバイス/負荷保護

・瞬停/熱/EMC対策



■ 小型化・軽量化

・携帯機器向け電源

・高密度サーバ用電源

・車載コンバータ・インバータ

■ 高機能化

・インテリジェントな電力管理(創・蓄・省エネ)

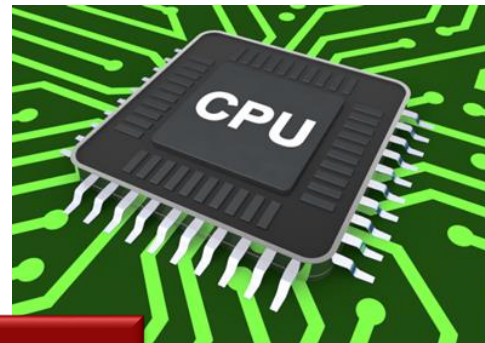
・通信による他システムとの連携

パワーエレクトロニクスとデジタル制御

- ◆ デジタル制御がパワーエレクトロニクスで徐々に浸透中

アナログを上回る効率性

ソフトによる柔軟性・高機能化



環境依存低減
(温度、劣化、部品ばらつき)

複数機能の1チップ集約

制御性能は制御手法/アルゴリズムの良し悪しに依存

モデルを活用したパワエレシステム開発

- ✓ シミュレーション: 試作前に様々な検証が可能、手戻り回避
- ✓ 実装/試験: 様々なハードウェアや試験環境に実装可能

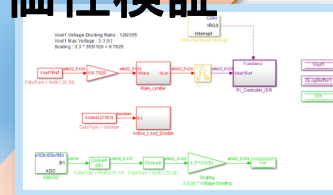
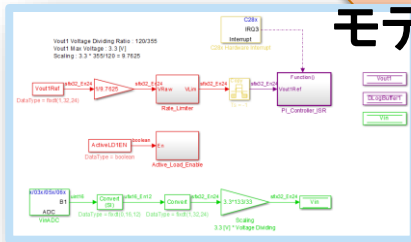
制御ソフト+回路
シミュレーション検証



制御ソフト実装
実回路検証

コード実装に向けた
モデル詳細設計

モデル&コード
等価性検証



自動コード生成

本講演でご紹介する内容

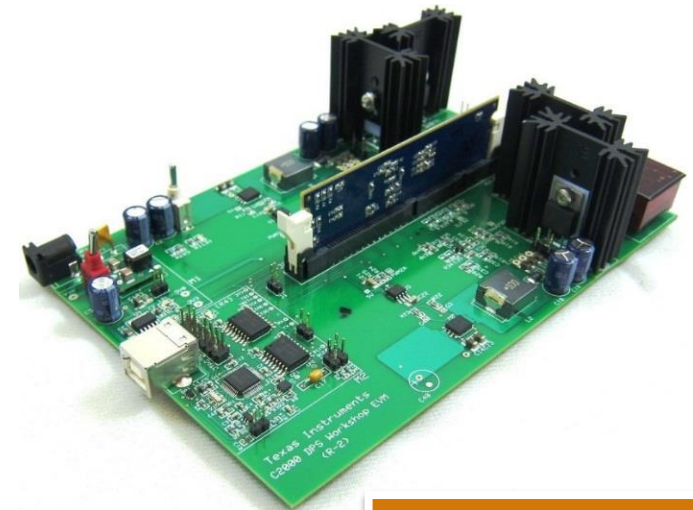
アジェンダ

- ソフトウェアへのアルゴリズム実装
- ハードウェアへのアルゴリズム実装
- 実機試験/他ツール連携ソリューション
- まとめ

実装のターゲット

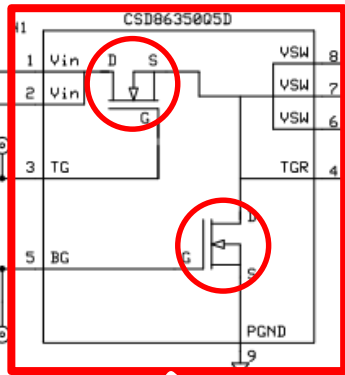
◆ Texas Instruments C2000 DC-DCコンバータトレーニングキット

- 搭載マイコン: F28035
 - 32bit CPU
 - 駆動周波数: 60MHz

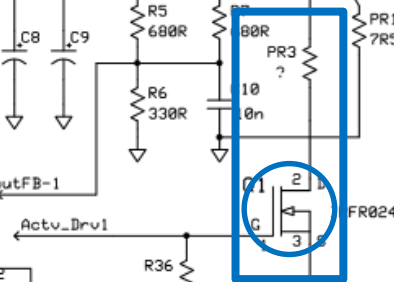


出力電圧
(目標電圧2V)

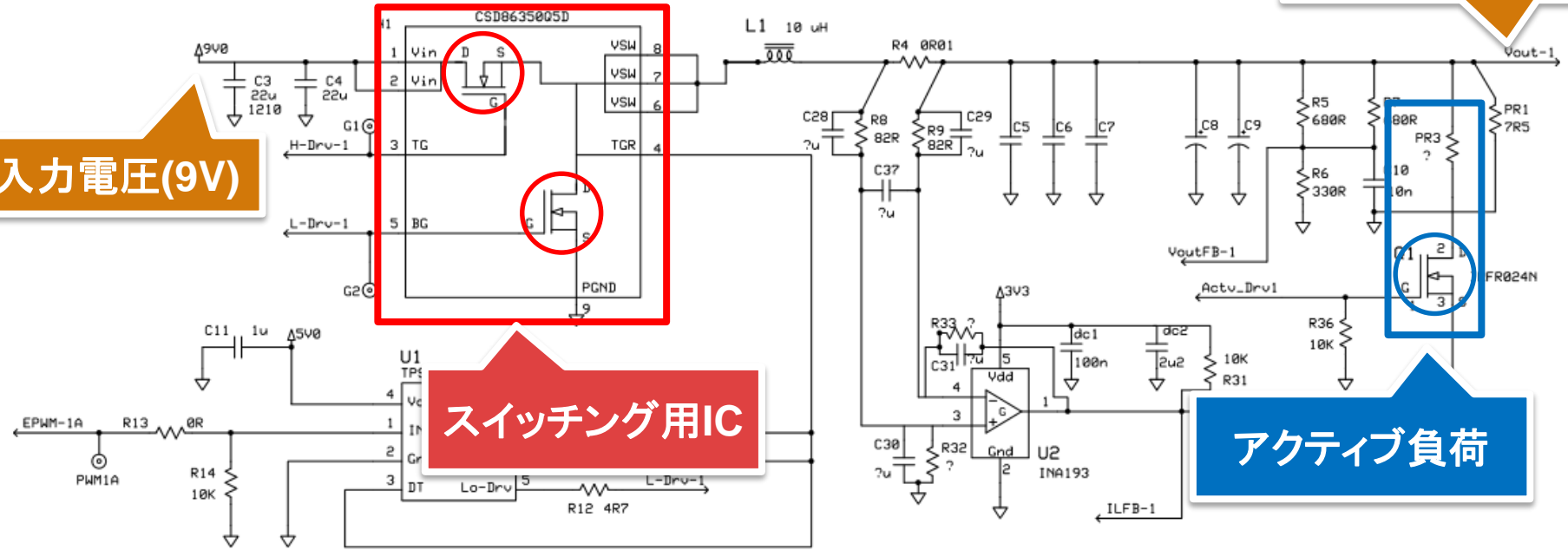
入力電圧(9V)



スイッチング用IC

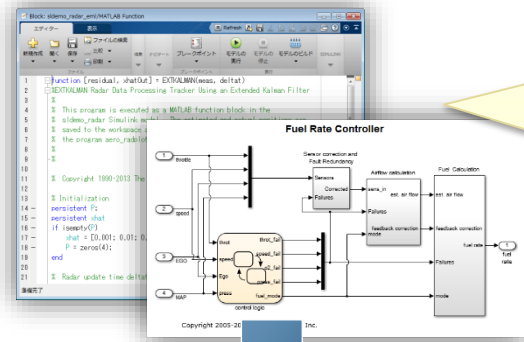


アクティブ負荷



モデル+Cコード自動生成ソリューション

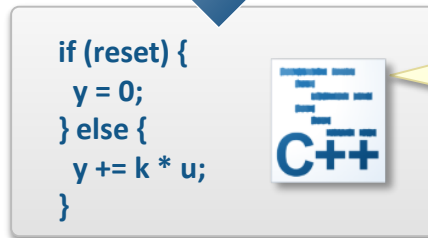
モデル



Simulink® / Stateflow®

- モデル作成・シミュレーション基本環境
- Fixed-Point Designer™**
- 固定小数点設計

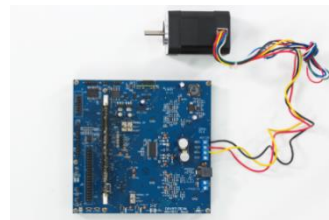
自動生成
コード



Embedded Coder®

- モデルから効率性・可読性に優れたC/C++コードを自動生成
- ※ライセンス上、MATLAB Coder, Simulink Coderも必要

既存コードと
組み合わせて利用



MCU/DSPに実装して
実験・量産試作

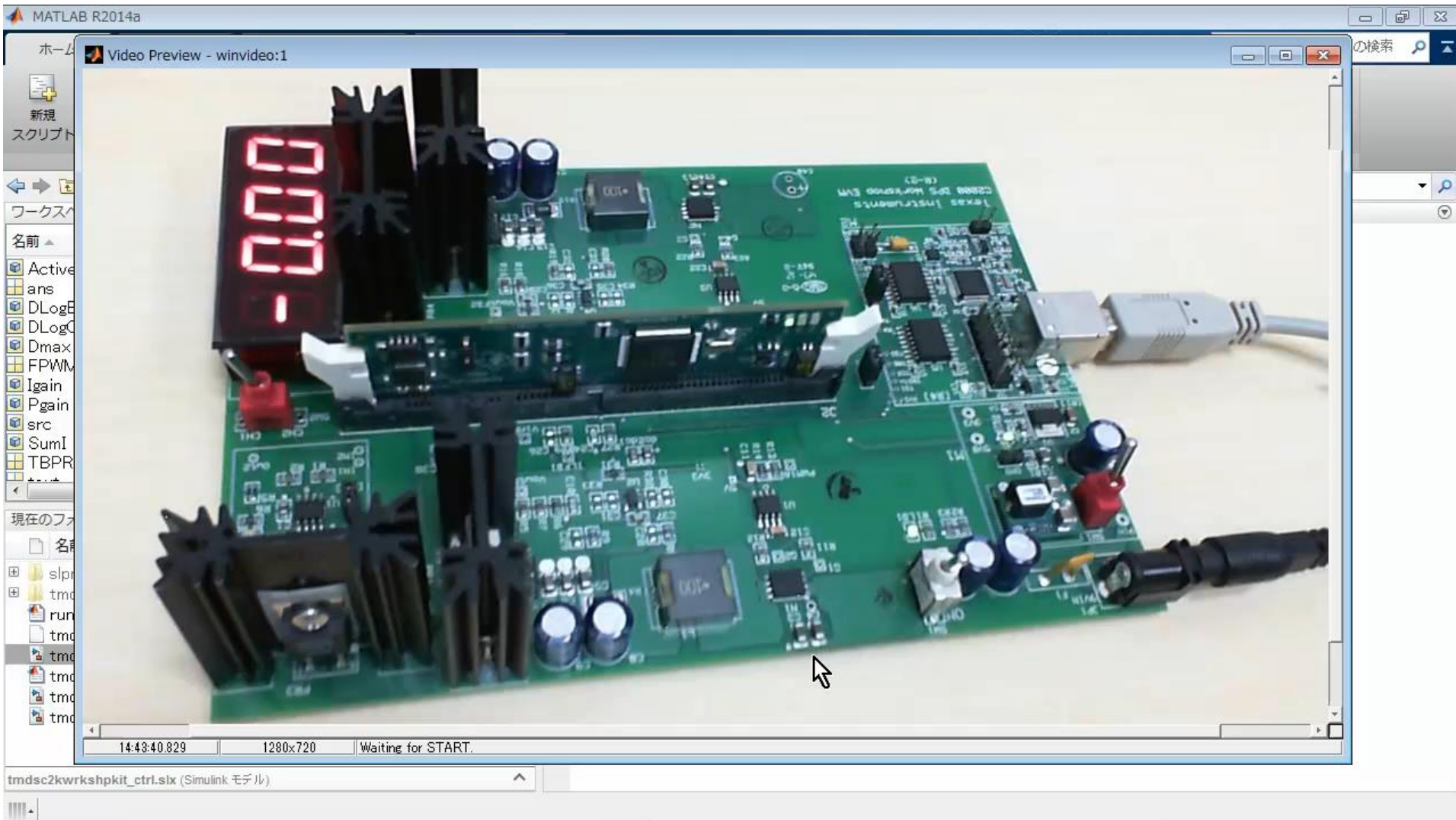
モデルをDLL化して
PC環境で利用



モデル生成コードの
利用は無料

モデルによるソフトウェア実装ダイジェスト

MOVIE



Embedded Coder モデル生成コードの利用タイプ



	ソフトコンポーネント型 (基本)	モデルビルド型 (応用)
利用イメージ	<p>アプリの一部をモデル化、生成されたコードを既存ソフトに組み込む</p>	<p>MCU/DSP用ブロックライブラリ利用、モデルを直接実装する</p>
主な用途	既存ソフトとの差分開発 HW/OS非依存アプリの開発	MCU/DSPを用いた制御実験 (On Target RCP)
特徴	標準ブロックのみ使用	専用ブロックライブラリを一部使用
I/O	グローバル変数 / 引数	AD/DA, PWM, CAN等のI/Oブロック
注意点	ビルド・結合は手動で行う必要アリ	対応MCU/DSP限定、HW知識も必要
対応 MCU/DSP	生成コードはANSI/ISO-C準拠なので ほぼ全てのMCU/DSPに対応可能	TI C2000, STMicro STM32F4 等 (MathWorksまたはベンダから提供)

ソフトウェア実装に向けたワークフロー

Embedded Coderによるコード生成&検証(モデルビルド型)

MATLAB&Simulink
Stateflow
Fixed-Point Designer

① 機能モデル開発

- ・ モデリング
- ・ シミュレーション/RCP検証

② 実装モデル開発

- ・ 実装用モデルリファクタリング
- ・ コンフィグ設定・チェック
- ・ 変数・定数・関数設計
- ・ 型・固定小数点設計

③ コード生成

- ・ Cコード自動生成

④ コード単体検証

- ・ コードレビュー
- ・ モデル-コードの等価性検証

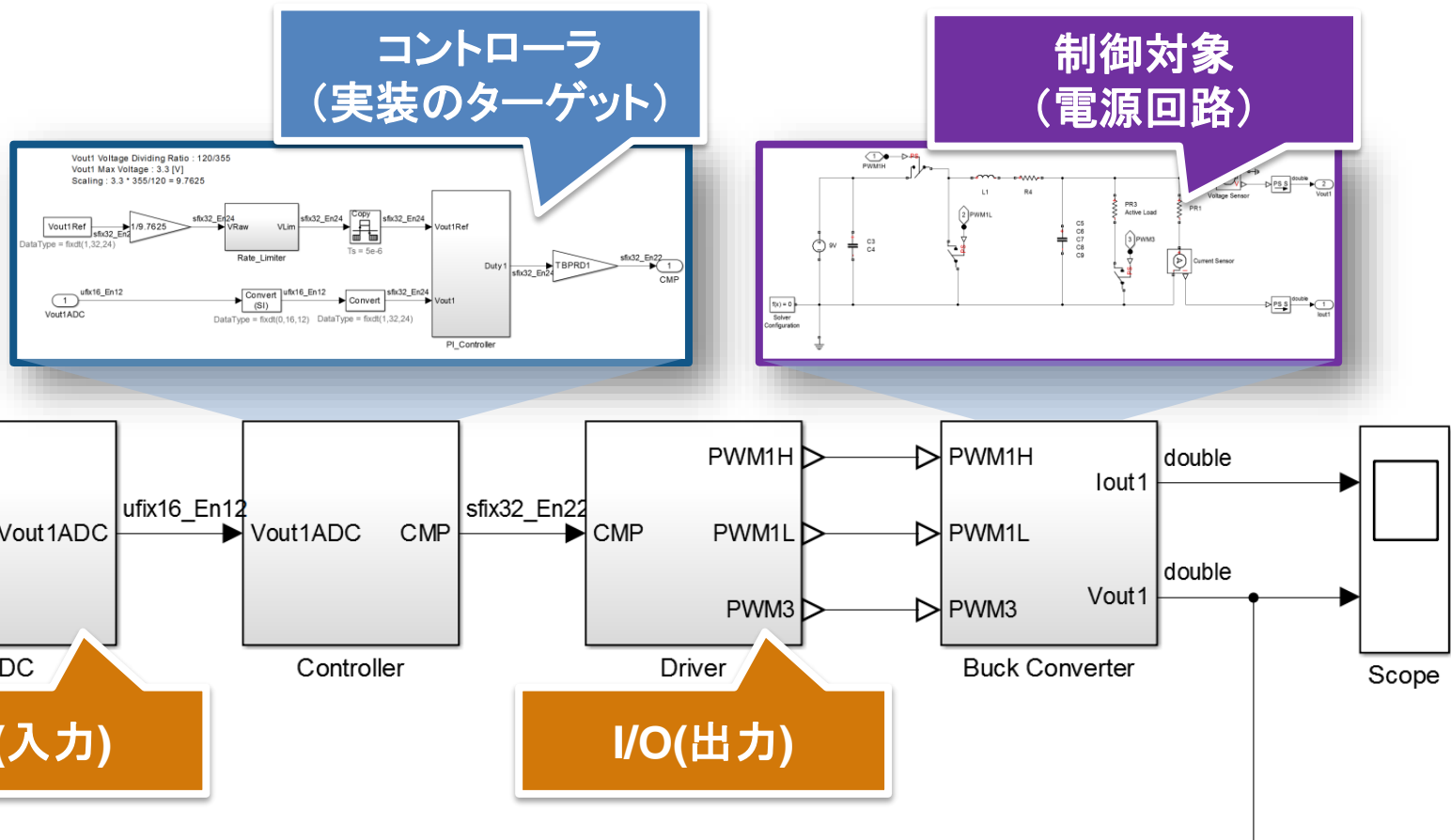
⑤ 実機評価

- ・ 実装・実験

Embedded Coder

DC-DCバックコンバータシステムモデル

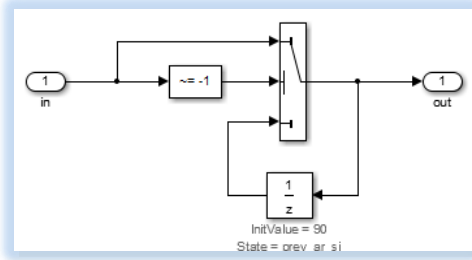
- ◆ コントローラ+電源回路をモデルで表現
- ◆ 制御アルゴリズムの性能をシミュレーションで評価



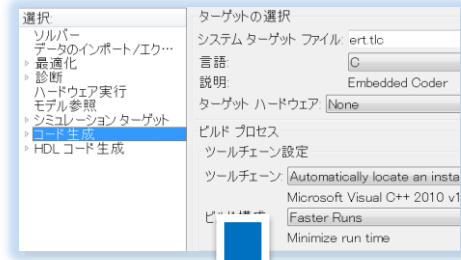
※電源回路はSimElectronicsを使用

自動コード生成に必要な情報

1 モデル
アルゴリズム



2 コンフィギュレーション
コード生成設定



3 データオブジェクト
データディクショナリ
変数・定数属性

Name	Value	Data Type	Min	Max
sw_di		uint8	[]	[]
pwmf_sao		uint8	[]	[]
pwmf_ao		uint8	[]	[]
state_snsr		Enum: EnumSNSR	[]	[]
state_ctrl		Enum: EnumCTRL	[]	[]
prev_ar_si		auto	[]	[]
sum_int		auto	[]	[]
timer		uint16	[]	[]
kp	0.1068	single	[]	[]
ki	0.0002	single	[]	[]
dz_v	0.5	single	[]	[]
bp	[-5 -0.02 0.02 5]	single	[]	[]
tbi	[-1 -1 1]	single	[]	[]

Embedded Coder

1 アルゴリズムの設計
2 コード生成関連の設定
3 変数・定数の属性定義
で自動生成されるコードが
形成されます

自動生成
Cコード

```

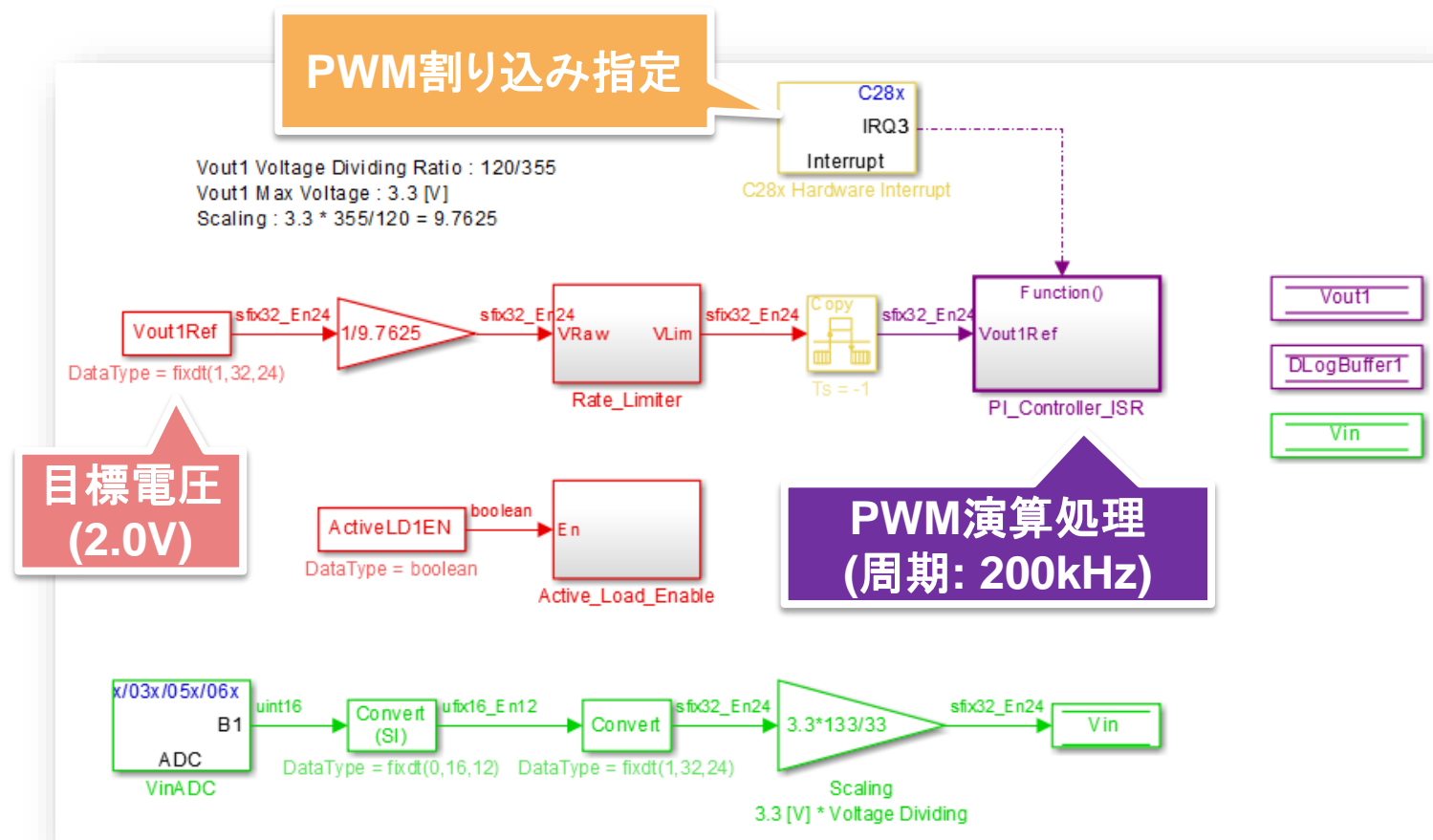
Hardware Interrupt Block: '

```

1 DC-DCバックコンバータ実装モデル

② 実装モデル開発

- ◆ システムモデルから実装対象(コントローラ)を取り出したモデル
- ◆ コントローラの入出力などをI/Oブロックに置き換え



2 コード生成のコンフィギュレーション

② 実装モデル開発

- シミュレーション(S) 解析(A) コード(C) ツール(T)
- ブロック線図の更新(U)
- モデルコンフィギュレーションパラメーター(F)



- ソルバー
- データのインポート/エクスポート
- 最適化
 - 信号とパラメーター
 - Stateflow
- 診断
 - ハードウェア実行
 - モデル参照
- シミュレーションターゲット
 - シンボル
 - カスタムコード
- コード生成
 - レポート
 - コメント
 - シンボル
 - カスタムコード
 - デバッグ
 - インターフェイス
 - 検証
 - コードスタイル
 - テンプレート
 - コード配置
 - データ型置換
 - メモリセクション
- HDLコード生成
 - Coder Target

ターゲットおよびビルドツールチェーンの指定

ターゲットの選択 システム ターゲット ファイル: ert.tlc 言語: C 説明: Embedded Coder ターゲット ハードウェア: TI Piccolo F2803x	ビルド プロセス ツールチェーン設定 ツールチェーン: Texas Instruments Code Composer Studio v5 (C2000) ビルド構成: Faster Runs Minimize run time
---	---

生成コードの最適化設定

シミュレーションとコード生成 <input checked="" type="checkbox"/> インライン パラメーター <input type="button" value="設定..."/>	<input checked="" type="checkbox"/> 信号ストレージの再利用
コード生成 <input checked="" type="checkbox"/> ローカルなブロックの出力を有効にする <input checked="" type="checkbox"/> 余分なローカル変数の削除 (式のたまたみ込み表現) グローバル データ アクセスの最適化: Use global to hold temporary results	<input checked="" type="checkbox"/> ローカル ブロック出力の再利用 <input checked="" type="checkbox"/> グローバル ブロック出力の再利用 <input checked="" type="checkbox"/> 配列のインデックスを簡略化する

コード置換の設定

ソフトウェア環境 標準の数学ライブラリ: C89/C90 (ANSI) コード置換ライブラリ: TI C28x 共有コードの配置: 共有場所
--

TI C2000モデルビルド型でのコード置換

◆ コード置換設定 & 専用ブロックで生成コードをカスタマイズ

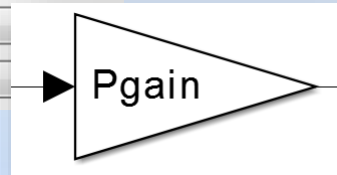
■ 設定によるコードの置き換え

ソフトウェア環境

標準の数学ライブラリ: C89/C90 (ANSI)

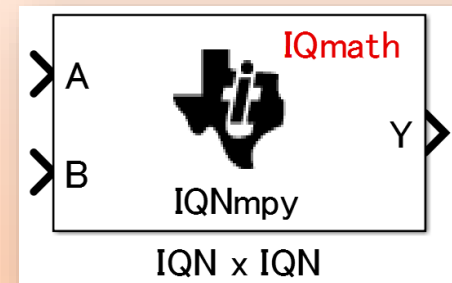
コード置換ライブラリ: TI C28x

共有コードの配置: 共有場所



```
/* Sum: '<S7>/Sum1' incorporates:
 * Gain: '<S7>/Gain'
 */
rtb_Gain = c28x_mul_s32_s32_s32_sr(Pgain, rtb_Gain_b, 24L) + SumI;
```

■ 専用ブロックによるコードの置き換え



```
/* C28x IQmath Library (stiiqmath_iqmpy) - '<S8>/IQN x IQN' */
{
  rtb_IQNxIQN = _IQ24mpy (rtb_Gain, Pgain);
}
```

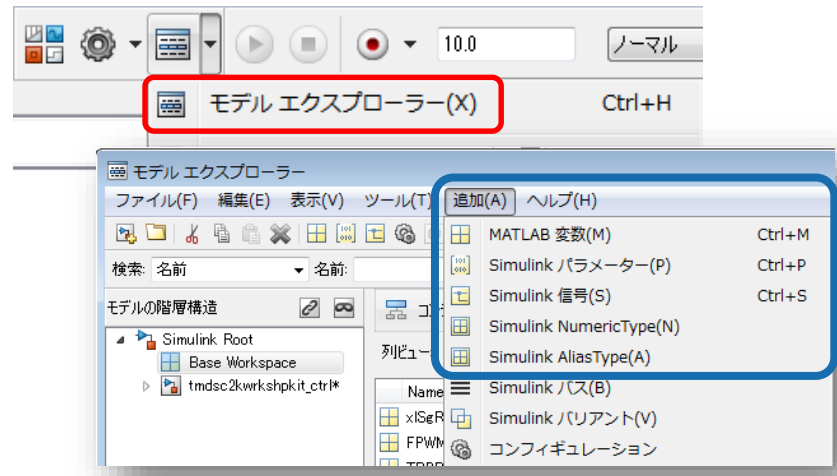
✓ TI C2000専用関数で演算処理の高速化を図れます

処理スピードはPILSによる実行プロファイルレポートで確認

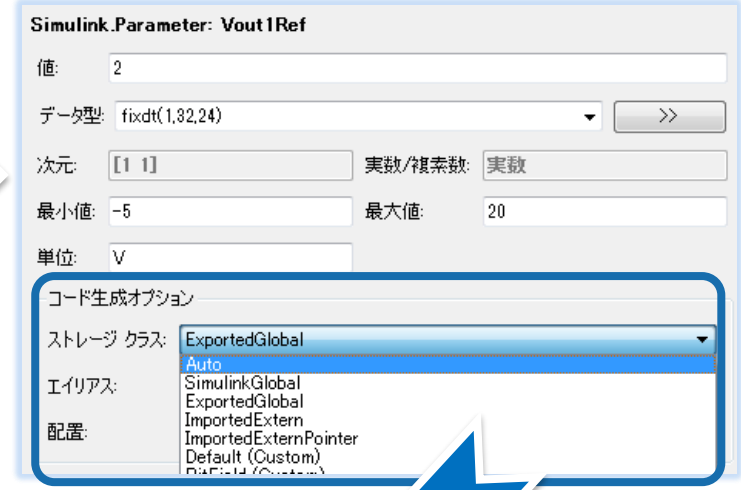
3 変数・定数 (RAM/ROM) 設計

コード生成時の変数の取り扱いをカスタマイズ

◆ 変数の定義



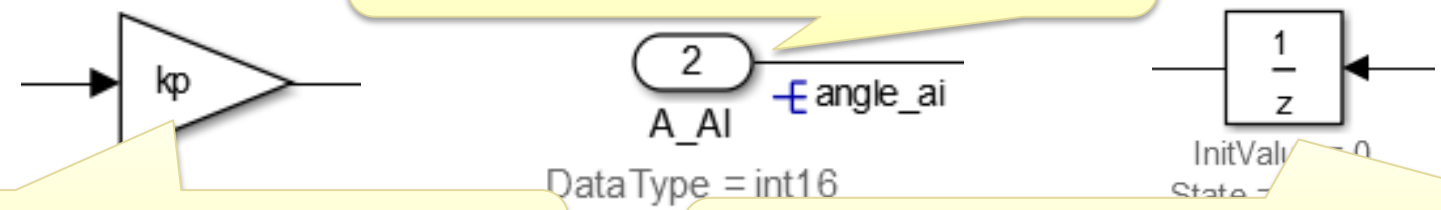
◆ データ型/宣言方法の指定



信号をグローバル変数として定義
`int16_T angle_ai;`

パラメータをマクロで定義
`#define kp 0.1068F;`

状態 (バッファ変数) に static 修飾子の付与
`static real32_T sum_int;`



3 Simulinkモデルの固定小数点設計

② 実装モデル開発

Fixed-Point Designerが固定小数点設計を強力サポート

出力の最小値: -5 出力の最大値: 100

出力データ型: fixdt(1,32,24)

データ型アシスタント

モード: 固定小数点 符号付き/なし: 符号付き 語長: 32

スケールリング: 2進小数点 小数部の長さ: 24

データ型オーバーライド: 継承 最高精度のスケールリングを計算

固定小数点の詳細

表現可能な最大値: 127.999999994039536

出力の最大値: 100

定数値: 2

出力の最小値: -5

表現可能な最小値: -128

精度: 5.960464477539063e-08

推奨スケールリング設定

詳細の更新

Vout1Ref sfix32_En24 → 1/9.7625

Data Type = fixdt(1,32,24)

固定小数点型 小数部長さ

符号付き 語長 スロープ

sfix32_En24

モデル全体のデータ型を一括オーバーライド

選択したシステムの設定

固定小数点の計測モード: ローカル設定を利用

データ型オーバーライド: ローカル設定を利用

データ型オーバーライド: double

ローカル設定を利用 スケールリングされた double

double

single

オフ

KP sfix16_En20 → KP double

各ブロックの語調・スケールリングを一括自動調整

専用GUI上でモデル内のブロックの最適な小数点設定を自動的に推定

Sum : Output	Run 1	<input type="checkbox"/>	n/a	fixdt(1,16,11)
Sum4 : Accumulator	Run 1	<input type="checkbox"/>	n/a	Taberit; Taberit
Sum4 : Output	Run 1	<input checked="" type="checkbox"/>	fixdt(1,16,3)	fixdt(1,16,5)
we	Run 1	<input type="checkbox"/>	n/a	fixdt(1,16,4)
we (base)	Run 1	<input type="checkbox"/>	fixdt(1,16,4)	fixdt(1,16,4)
we_ref	Run 1	<input type="checkbox"/>	n/a	fixdt(1,16,4)

選択したシステムのオートスケールリング

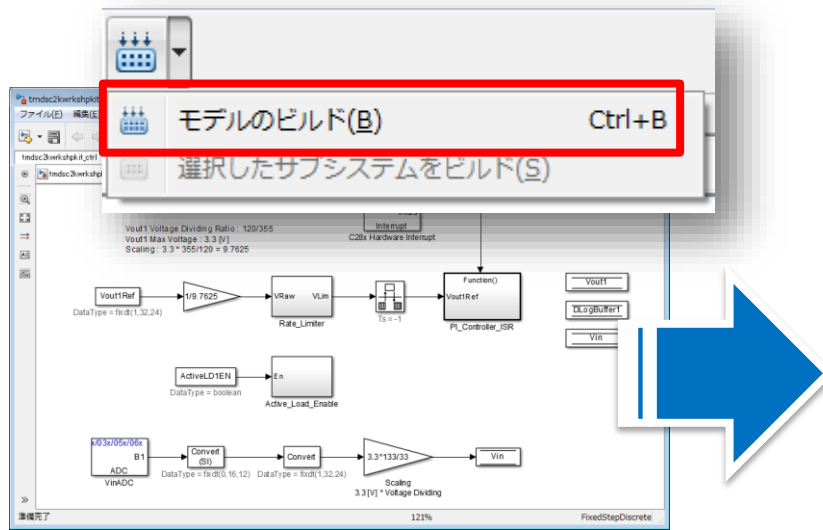
DT 小数部の長さを推奨 設定...

 選択した小数部の長さを適用

コード生成 & コードレビュー

③ コード生成

◆ 諸設定が済んだ後はビルドボタン一つでコードを生成できます



コードメトリクス
(行数、複雑性、RAMサイズ等)

3. Function Information

View function metrics and estimated stack size of subroutines that the function calls.

View: Call Tree | Table

Function Name	Accumulated Stack Size (bytes)	Self Stack Size (bytes)	Lines of Code	Lines Complexity
[+] controller_pi_step	28	14	63	144
[+] controller_pi_initialize	22	0	10	26
controller_pi_terminate	0	0	0	4

Code Generation Report

Contents

- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Code Replacements Report

Highlight Navigation

Previous Next

```

86 extern void init_HRPWM ( volatile struct EPWM_REGS * EPwmRegs, uint16_T HRLC
87 uint16_T CTLMODE, uint16_T EDGMODE, uint16_T CMPAHR, uint16_T TBPFSHR,
88 uint16_T TBPFSHR,
89 uint16_T HRPREDx, uint16_T AUTOCONV, uint16_T HRUPDOWN);
90 extern void update_MapScaleFactor(void);
91 void isr_int3pie1_task_fcn(void);
92 void isr_int3pie1_task_fcn(void);
93 static void rate_monotonic_scheduler(void);
94
95
96 /* Hardware Interrupt Block: '

```

生成ファイルリスト

- Generated Code
- [-] Main file
 - ert_main.c
 - [-] Model files
 - tmdsc2kwrkshpk_it_ctrl.c (2)
 - tmdsc2kwrkshpk_it_ctrl.h
 - [-] Shared Utility files
 - const_params.c
 - mw_C28x_s32.h

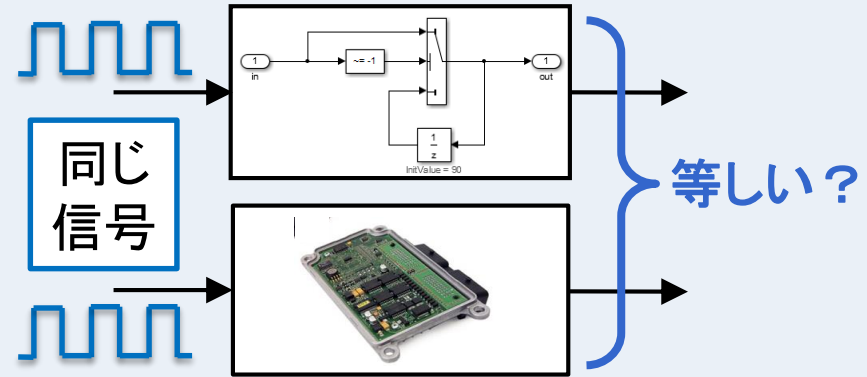
コード・モデル間リンク

※モデルビューが無い場合、モデルと直接リンクされます

等価性検証&パフォーマンステスト

- ◆ 処理系依存動作やコード生成設定不備による問題をチェック

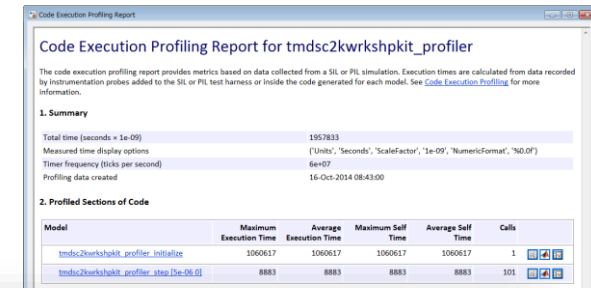
PILS: **P**rocessor **I**n the **L**oop **S**imulation
 マイコン/マイコンシミュレータ上で
 生成コードを動作(ターゲット検証)



- ✓ PILSの場合演算負荷の測定が行えます

使用ブロック / 最適化設定	演算時間
TI専用ブロック	2.317μsec
通常ブロック(コード置換あり)	3.517μsec
通常ブロック	8.883μsec

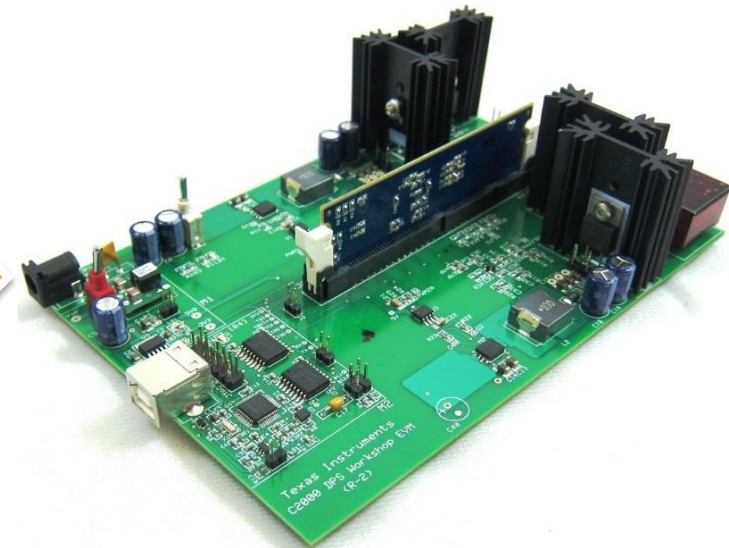
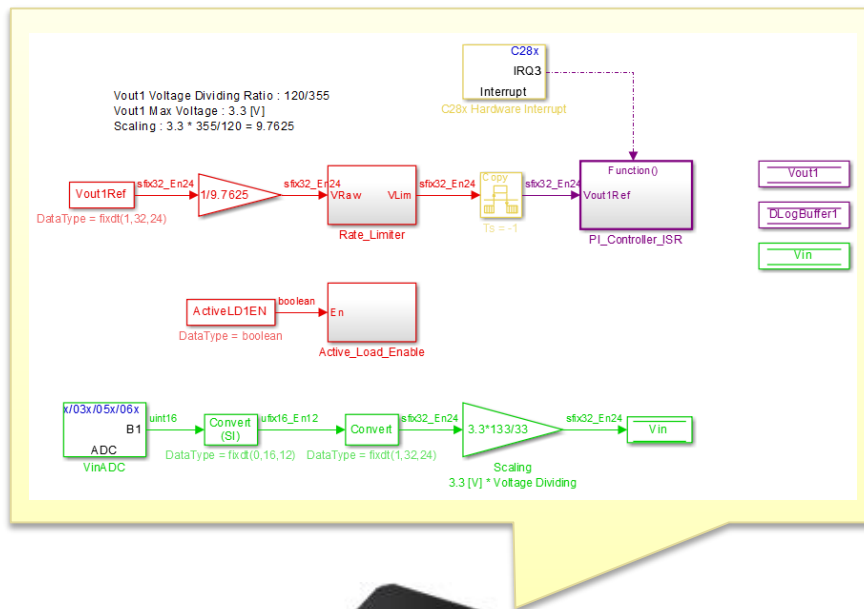
コード置換された処理なら
 PWM一周期(5μsec)内で演算可能!



Model	Maximum Execution Time
tmdsc2kwrkshpkit_profiler_initialize	1060617
tmdsc2kwrkshpkit_profiler_step [5e-06 0]	8883

実機試験

- ◆ 実行ファイルはビルド時にMCUへ自動的にダウンロード
- ◆ モデルのデザイン完了後、すぐに実機試験へ移れます

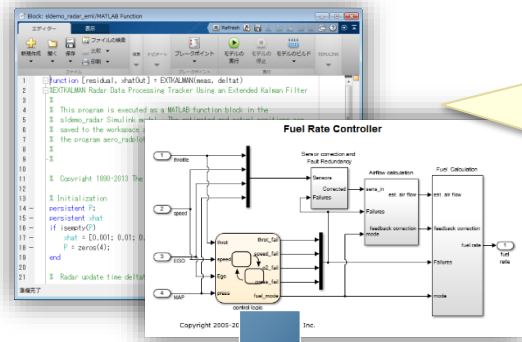


アジェンダ

- ソフトウェアへのアルゴリズム実装
- ハードウェアへのアルゴリズム実装
- 実機試験/他ツール連携ソリューション
- まとめ

モデル+HDLコード自動生成ソリューション

モデル



Simulink[®] / Stateflow[®]

- モデル作成・シミュレーション基本環境

Fixed-Point Designer[™]

- 固定小数点設計

自動生成
コード

```
always @(posedge CLK) begin
  if (QL==4'd9)
    QL <= 4'd0;
  else
    QL <= QL + 1'b1;
  end
end
```

**VHDL,
Verilog**

HDL Coder[™]

- HDLコード(VHDL/Verilog)の生成

HDL Verifier[™]

- HDLコードの検証機能の提供

既存コードと
組み合わせて利用



FPGA/ASICに実装して
実験・量産試作

ハードウェア実装に向けたワークフロー

HDL Coder / HDL Verifierによるコード生成 & 検証

MATLAB&Simulink
Stateflow
Fixed-Point Designer

① 機能モデル開発

- モデリング
- シミュレーション

② 実装モデル開発

- 実装用モデルリファクタリング
- コンフィグ設定・チェック
- 変数・定数・関数設計
- 型・固定小数点設計

HDL Coder

③ コード生成

- HDLプロパティ設定
- HDL生成

HDL Verifier
(ModelSim, Incisive)

④ HDL検証

- コードレビュー
- Co-SimulationによるHDL検証
- HDL合成
- FILによるHDL検証

⑤ 実機評価

- FPGAボードスタンドアロン実装
- 実装・実験

HDL Coderによるコード生成

- ◆ Cコード生成同様、モデルから直接HDLコードを生成
- ◆ レポートのトレーサビリティ機能でコードレビューもスムーズ

The screenshot displays the HDL Coder environment. On the left, a Simulink model for a buck converter controller is shown. It includes blocks for gain calculation (Pgain, Igain), reference signals (Vout1Ref, Vout1SlewRate), and a central controller block (Buck_Converter_Controller). A red circle highlights the 'uint16' data type of the Vout1ADC input. A large blue arrow points from the model to the Code Generation Report window on the right.

The Code Generation Report window shows the following code snippet:

```

61 // <SI>/Gain1
62 assign Gain1_mul_temp = 1718537 * Vout1Ref;
63 assign Gain1_out1 = Gain1_mul_temp[48:24];
64
65
66
67 // <SI>/Rate Limiter
68 Rate_Limiter u_Rate_Limiter (.clk(clk),
69 .reset(reset),
70 .enb_1_200_0(enb_1_200_0),
71 .VRaw(Gain1_out1), // sfix25_En24
72 .Vout1SlewRate(Vout1SlewRate), // sfix32_En24
73 .VLim(Rate_Limiter_out1) // sfix25_En24
74 );
75
76 // <SI>/Rate Transition2
77
78

```

Below the code, the report lists generated source files:

- Integrator.v
- PI_Controller.v
- Rate_Limiter.v
- Buck_Converter_Controller.v (1)**
- buck_cn timer.v
- buck_cn timer.v

The bottom right of the report window shows a block diagram of the Rate_Limiter block, illustrating the internal signal flow and logic.

ハードウェア実装ソリューションの主な機能

◆ HDL Coder

ターゲット非依存のVHDL/Verilog生成

- ✓ VHDL(IEEE1076-1993)
- ✓ Verilog(IEEE1364-2001)

```

-- <S14>/Gain
Gain_mul_temp <= 5094059 * In1_signed;
Gain_out1 <= Gain_mul_temp(46 DOWNTO 23);
    
```

```

// <S14>/Gain
assign Gain_mul_temp = 5094059 * In1;
assign Gain_out1 = Gain_mul_temp[46:23];
    
```

面積と速度間のトレードオフの調整

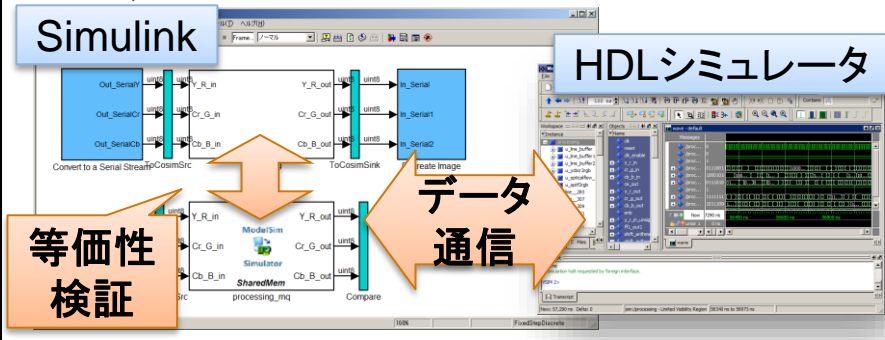
- ✓ パイプラインの追加とリソース共有の設定
- ✓ クリティカルパスの表示

- 遅延の調整
- 分散型パイプライン方式の優先順位 Numerical Integrity ▼
- 階層分散型パイプライン方式
- 設計遅延の保持
- タイミングコントローラーの最適化

◆ HDL Verifier

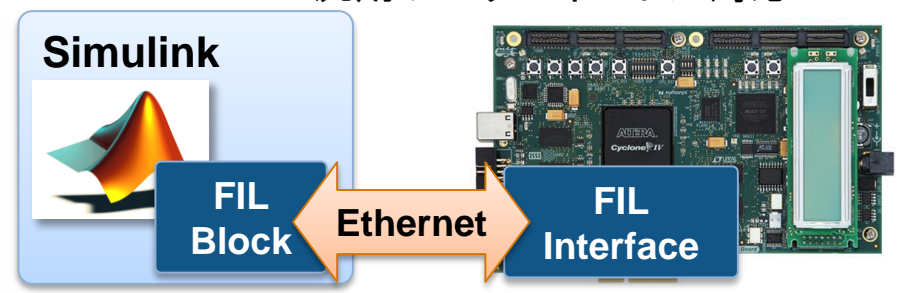
HDL開発ツールとの協調シミュレーション

- ✓ 実装時の挙動をシミュレータで検証



FPGA in the Loopによる検証

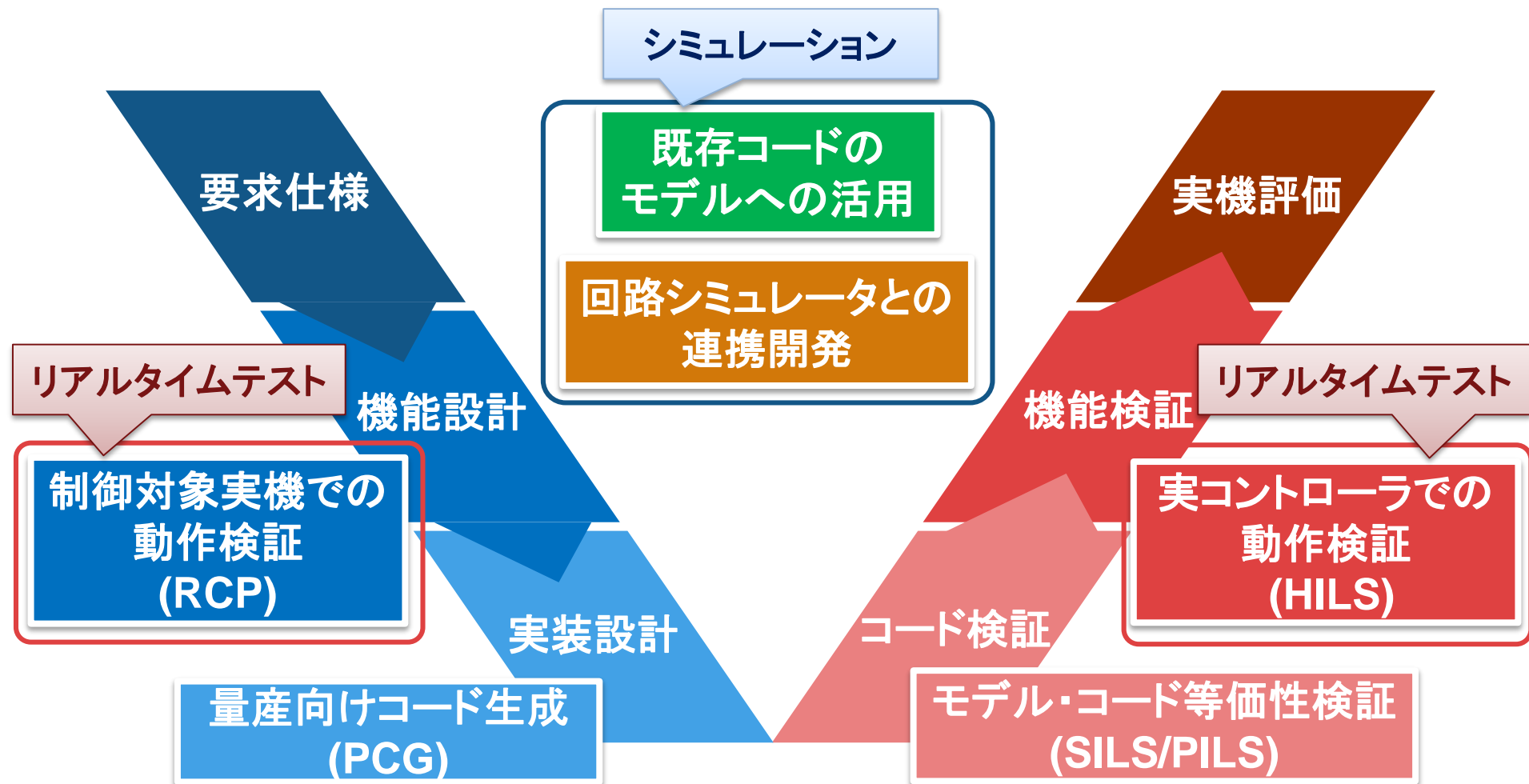
- ✓ FPGAボードによる高速プロトタイプ検証
- ✓ Xilinx・Altera汎用/カスタムボードに対応



アジェンダ

- ソフトウェアへのアルゴリズム実装
- ハードウェアへのアルゴリズム実装
- 実機試験/他ツール連携ソリューション
- まとめ

実機試験/他ツール連携ソリューション



RCP: Rapid Control Prototyping
PCG: Production Code Generation

SILS: Simulation In the Loop Simulation
PILS: Processor In the Loop Simulation
HILS: Hardware In the Loop Simulation 47

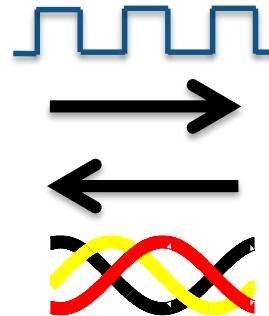
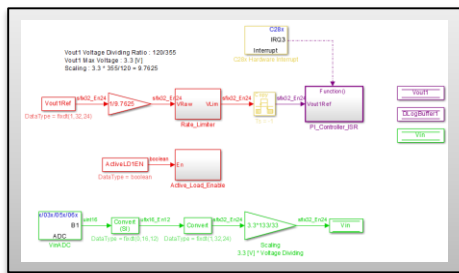
リアルタイムテストソリューション

RCP (Rapid Control Prototyping)

- ✓ 実装基板完成まで待てない
- ✓ モデルでスグに実機を動かしたい

マイコン向け設定無しで制御モデルを即座に実行 & 実制御対象でテスト

コントローラーモデル



制御対象
実機



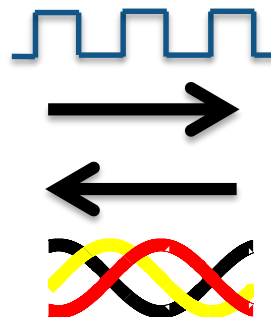
HILS (Hardware In the Loop Simulation)

- ✓ 本番の実機試験に向けた事前チェック
- ✓ 再現が難しいテストをしたい

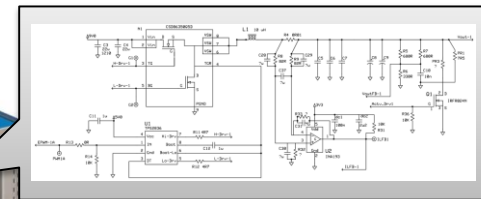
制御対象を模擬したシミュレーション環境でコントローラ実機を詳細テスト



コントローラ
実機

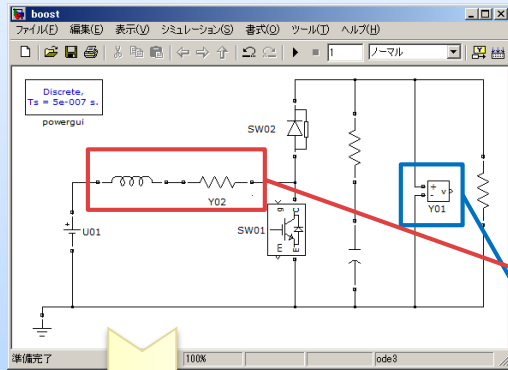


制御対象

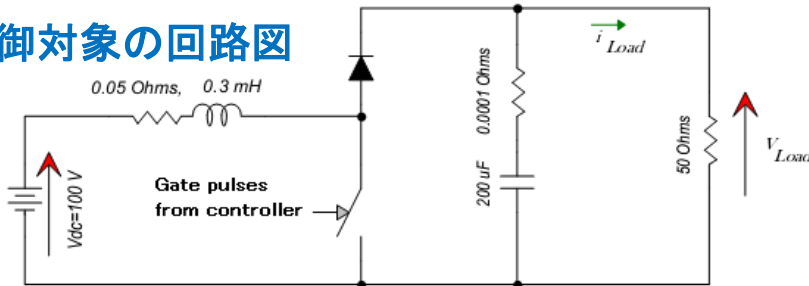


デジタル電源制御コントローラ向けHILS

SimPowerSystems™を用いて
回路モデルを作成



制御対象の回路図

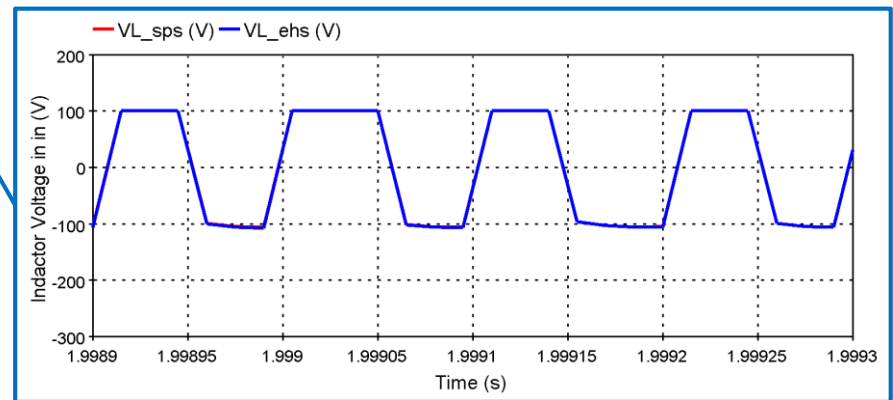
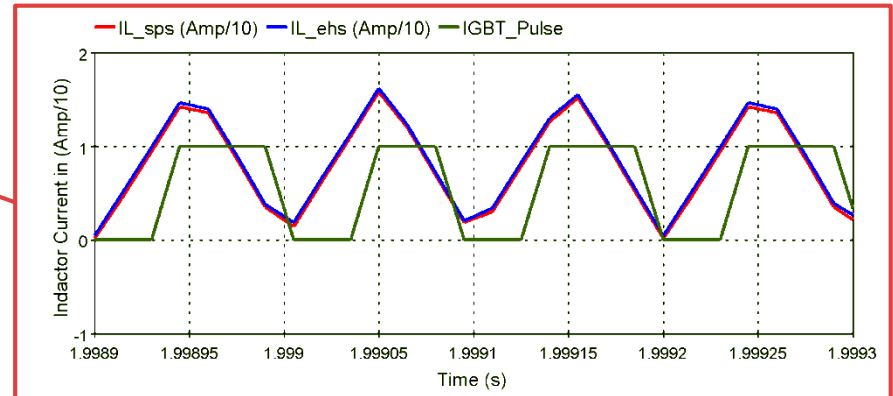


FPGA上に回路構成を再現



OPAL-RT社RT-LAB(リアルタイムシミュレータ)
+ eHSソルバ

SimPowerSystems™を用いた
シミュレーション結果(500ns, Tustin)と
ほぼ同じ精度でHILS環境の構築が可能



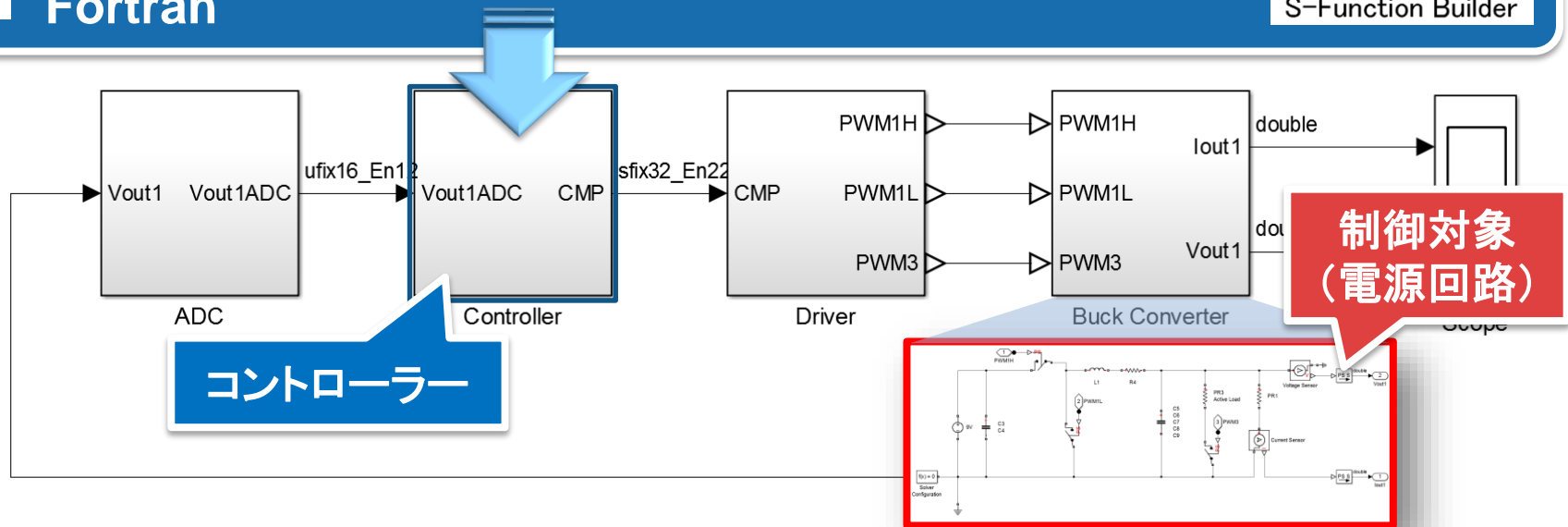
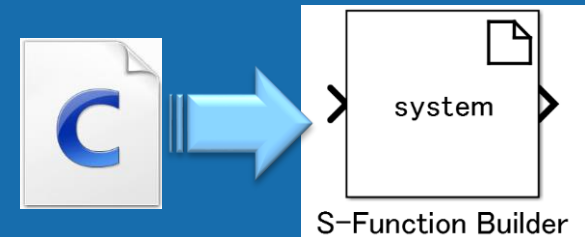
RT-LAB + eHS、SimPowerSystems™それぞれのシミュレーション結果
(※DC電圧:100V, PWMキャリア周波数:10KHz, Duty:50%)

既存ソースコードのモデルへの活用

- ◆ コード資産をスムーズにSimulinkモデルで活用できます

既存ソースコード取り込み用ライブラリブロック

- MATLABスクリプト
- C言語/C++言語
- Fortran



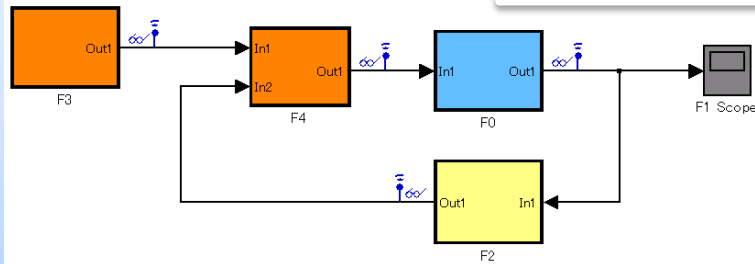
- ✓ ハンドコードのアルゴリズムをシステムレベルで動作検証
- ✓ 既存コードから徐々にモデルによる開発へシフトが可能

アナログ・ミックスシグナル回路設計へのモデルの活用

HDL VerifierによるSystemVerilog DPI生成

System Design

Simulink



Cコード生成機能により
ビヘイビアモデルとしてエクスポート



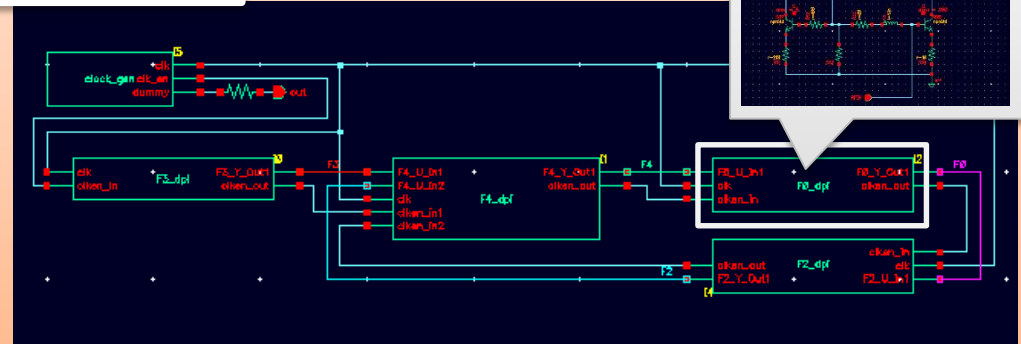
Circuit Design

Cadence® virtuoso® AMS Designer

システムモデルの活用で詳細回路の検証
& 検証の効率化によるTATの削減

回路シミュレータ上で
詳細モデルの作り込み

- システムモデル活用で設計回路の早期システム検証
- システム等価検証に活用
- 複雑な信号で検証可能

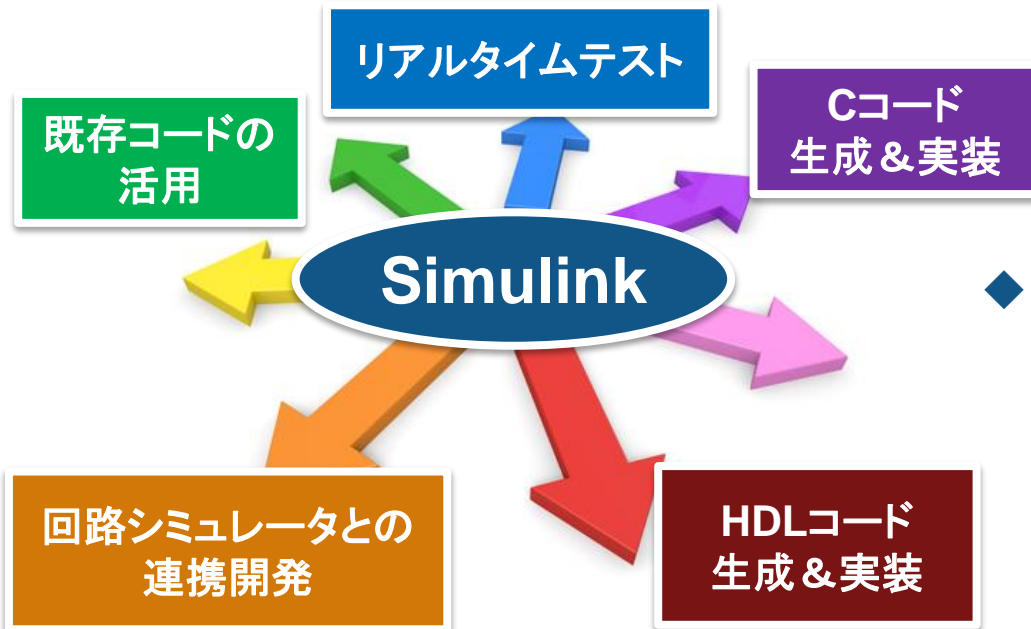


アジェンダ

- ソフトウェアへのアルゴリズム実装
- ハードウェアへのアルゴリズム実装
- 実機試験/他ツール連携ソリューション
- まとめ

まとめ

- モデルによるアルゴリズム開発 & コード生成ソリューションでシームレスな開発フローを実現
- 素早い修正ループでより本質的な開発業務にリソースをシフト



- ◆ モデル資産を柔軟に量産実装・試験まで有効活用



Accelerating the pace of engineering and science

ご清聴ありがとうございました

© 2014 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.